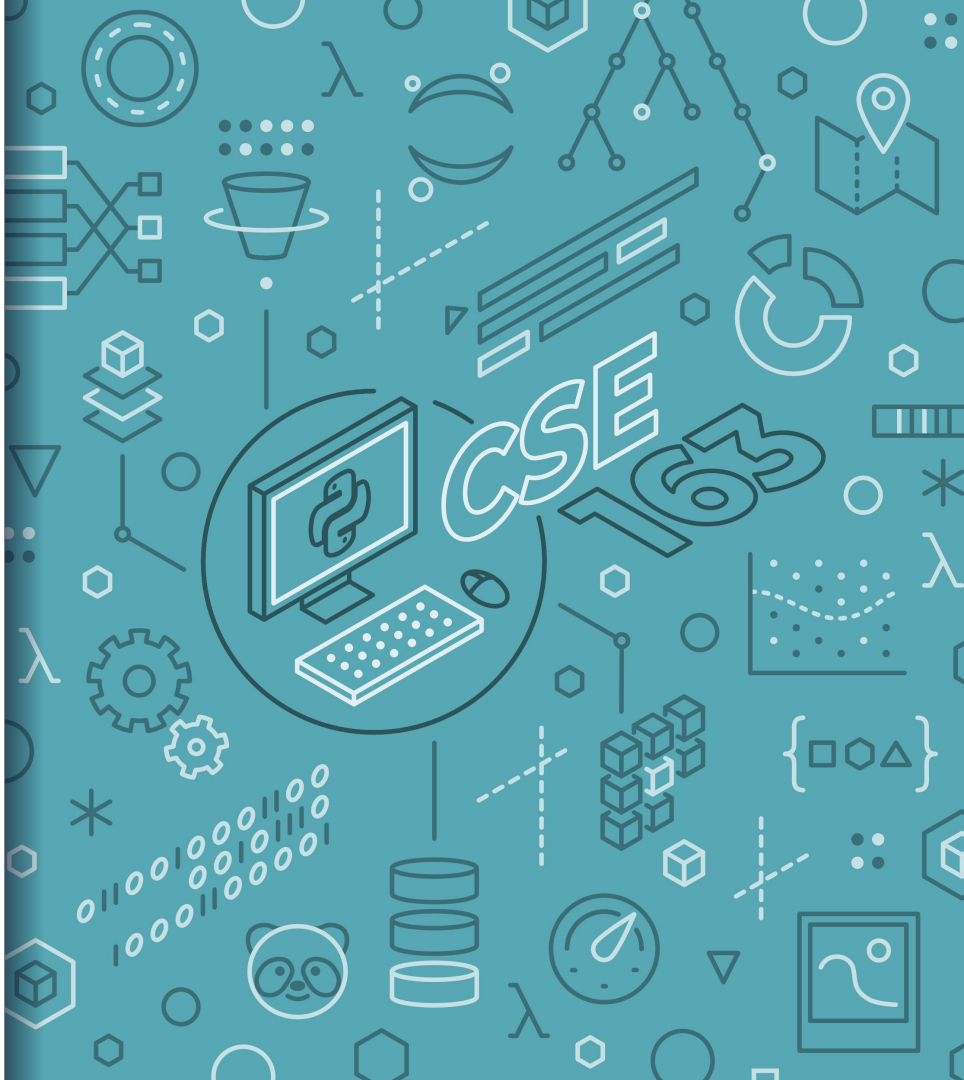


CSE 163

Introduction to Python II

Hunter Schafer

Last Time Time

- Overview of class
- Syllabus
- Python Crash Course - Day 1
  - Hello world
  - Variables
  - Expressions

This Time

- Jupyter Notebooks
- Python Crash Course - Day 2
  - Types + Casting
  - Loops
  - Conditionals
  - Functions (parameters + returns)
  - Strings

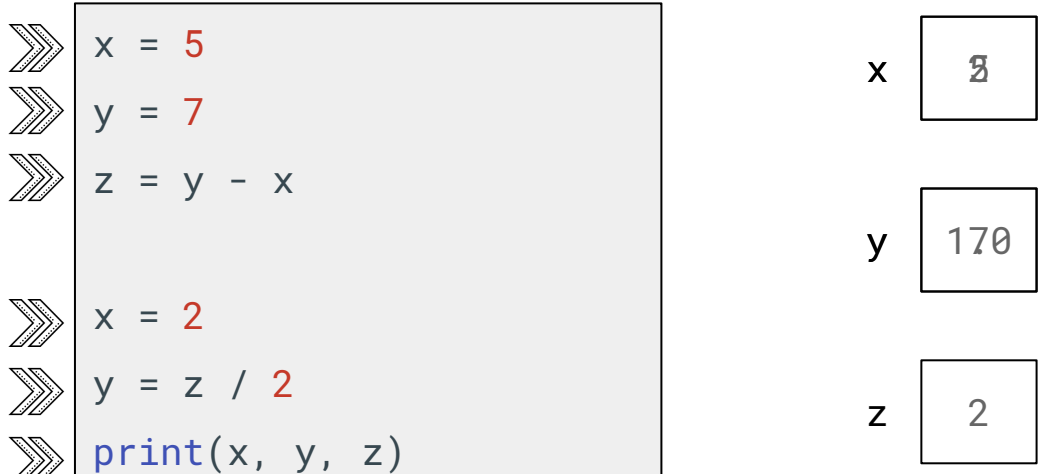
Poll Everywhere

Make a prediction of what this program will output.

Note that `print(1, 2, 3)` outputs: `1 2 3`

```
x = 5
y = 7
z = y - x

x = 2
y = z / 2
print(x, y, z)
```




```
> Output

2 1.0 2
```

Python Modes

- Script
  - Write a .py file and run it
    `python my_file.py`
    Your HW + Project
  - Runs file from top to bottom
- Interactive Shell
  - **R**ead **E**valuate **P**rint **L**oop (**REPL**)
  - Interactive mode to try small chunks of code
  - Simple: `python`
  - Complex: Jupyter Notebook
    Lecture (usually)
- Follow Software instructions to set up your computer for development
- For those of you following on your laptops, you can play around with the Notebook I'm using by using this link

bool
True/False

- A `bool` has two values: `True`, `False`
- Can you logical operators: `and`, `or`, `not`

```
b1 = False
b2 = True
print(b1 and b2)       # False
print(b1 or b2)        # True
print(not b2 or b1)    # False
print(not (b2 or b1))  # False
```

- Can get `bool`s by comparing numbers

```
x = 3
print(x < 4)  # True
print(x >= 5) # False
print(x == 2) # False
```

Casting + Types

- Every piece of data in Python has a **type**. You can convert between types by **casting**.

```
x = 1.4
print(x)          # 1.4
print(int(x))     # 1

x = "1.7"
print(float(x))   # 1.7
print(int(x))     # Error
```

- Commonly used types: `int`, `float`, `bool`, `str`
- Can use `type(x)` to find x's type

While Loop

- A **while loop** has a **condition** and a **body**. It executes the body repeatedly until the condition is false.

```
x = 1
while x < 100:
    print(x)
    x = x * 2
print('After loop', x)
```

```
1
2
4
8
16
32
64
After loop 128
```

- **Important**: Python uses indentation to determine what belongs inside the loop!
  - Very common beginner error
    `IndentationError: unexpected indent`

For loop

- If you know Java, for loops in Python look pretty different
- A for loop has a **body** that repeats for every value in a **sequence**, using a **loop variable** to track the current value

```
for i in range(5):
    print(i)
print('After loop', i)
```

```
0
1
2
3
4
After loop 4
```

- The many uses of `range`
  - `range(A)`
    - Between 0 inclusive and A exclusive
  - `range(A, B)`
    - Between A inclusive and B exclusive
  - `range(A, B, C)`
    - Between A inclusive and B exclusive (steps by C)

Brain Break

Conditionals

- You can use conditional statements to have your program execute different **branches** of code based on **test conditions**
  - Keywords: `if / elif / else`

```
x = 14
if x < 10:
    print('A')
elif x >= 13:
    print('B')
else:
    print('C')
```

- **Q: What values of x fall into the else case?**
  - `x >= 10 and x < 13  # 10, 11, 12`

Poll Everywhere

Think

1 minute

pollev.com/cse163

Make a prediction of what this program will output.

```
for i in range(1, 10, 3):
    if i % 2 == 0:
        print('A')
    elif i >= 5:
        print('B')
    print('C')
```

Options (different lines of output separated by "/")

- C / A / B / A
- C / A / C / B / C / A / C
- C / A / B
- C / A / C / B / C

Poll Everywhere

Pair

2 minutes

pollev.com/cse163

Make a prediction of what this program will output.

```
for i in range(1, 10, 3):
    if i % 2 == 0:
        print('A')
    elif i >= 5:
        print('B')
    print('C')
```

Options (different lines of output separated by "/")

- C / A / B / A
- C / A / C / B / C / A / C
- C / A / B
- C / A / C / B / C

Calling Functions

- A **function** is a named procedure with a series of instructions that can be **called** in your program to execute those instructions
  - To call a function: `function_name()`
- Python has many **built-in** functions that you can call you can use in your programs. Here are some examples:
  - `print`
  - `range`
  - Casting: `int, float, bool, str`
  - Math: `abs, max, min, sum`
- A function can take **arguments** to pass information to the function and may **return** values

```
x = max(5, min(4, 7))  # x will be 5
```

Defining your own Function

- Jupyter Notebook Demo

```
def greeting():
    print('hello!')
```

- Define arguments between the parens
- Use return statements to return values

```
def mean(a, b):
    print('Calling mean with', a, b)
    return (a + b) / 2

def main():
    x = mean(4, 5)
    print(x)

if __name__ == '__main__':
    main()
```

Strings

- `str` that represents a sequence of characters
- See Jupyter Notebook Demo for reference on how to manipulate strings. Important concepts covered in these notes
  - How to define a string
  - String concatenation
  - Index into strings
  - Length of string & How to loop over a string
  - String methods: split, upper, lower
    - Loop over a list

Fancy String Indexing

- You can index into strings (or lists) by passing in more than one number as a **slice**

```
s = 'hello world'

s[0]              # h
s[0:len(s)]       # hello world
s[2:]             # llo worl
s[:len(s)-1]      # hello worl
s[2:len(s)-1]     # llo worl
s[2:len(s)-1:2]   # lowr

s[-1]             # d
s[::-2]           # drwolh
```

- See guide here for a longer explanation

Next Time

- Documenting code
- Testing
- Lists
- Files

Before Next Time

- Go to section tomorrow
- Finish Python setup!